Conceitos básicos de Tkinter

O Tkinter é uma das ferramentas que o Python oferece para o desenvolvimento de interfaces gráficas. Uma vantagem do Tkinter é que a instalação do pacote básico de Python para Windows já incorpora o Tkinter, ou seja, qualquer computador que tenha o interpretador de Python instalado já permite criar interfaces gráficas em Tkinter (algumas distribuições em Linux requerem a instalação de módulos especiais).

Para importar o tkinter:

```
import tkinter as tk
```

```
from tkinter import *
```

Para criar uma janela GUI, o Tkinter fornece a classe Tk() e é dela que conseguimos extrair todos os *widgets* com os quais construiremos a interface. Os botões de minimizar, maximizar e fechar já são automaticamente colocados na janela, além do título e do símbolo característico do Tkinter.

O método *mainloop* faz com que a janela GUI fique aberta esperando pelo acontecimento de eventos até que ocorra algum evento que a "destrua", como um clique no botão X de fechar ou num botão "Fechar" que eventualmente possa existir dentro da janela principal. O *mainloop* deve ser sempre chamado como a última linha lógica de código do seu programa.

```
import tkinter as tk
main_window = tk.Tk()
main_window.mainloop()
```

Alternativamente, a interface gráfica pode ser estruturada sob a forma de classes, e uma instância da classe Tk, que no exemplo chamamos main_window, terá que ser sempre passada como argumento do método construtor da janela.

```
from tkinter import *
class Janela:
      def __init__(self, instancia_de_Tk):
            pass
main_window=Tk()
Janela(main_window)
main_window.mainloop()
```

Para alterar o título da janela, usa-se a função *title* na janela principal ou então o método classname ao declarar a classe Tk().

```
import tkinter as tk
main_window = tk.Tk()
main_window.title('Python GUI New Title')
main_window.mainloop()
```

```
import tkinter as tk
main_window = tk.Tk(className='Python GUI New Title')
main_window.mainloop()
```

Para definir o tamanho da janela, usa-se a função *geometry* na janela principal.

```
import tkinter as tk
main_window = tk.Tk()
main_window.geometry('350x200')
main_window.mainloop()
```

Depois de criar uma janela GUI usando a classe Tk() e antes de chamar a função mainloop(), é possível adicionar os *widgets* que forem necessários.

Após a definição de um *widget*, é necessário usar um gerenciador de geometria para indicar em que posição o *widget* aparecerá dentro da janela principal. O Tkinter oferece três opções: grid, pack e place. Neste documento vamos abordar o pack e o grid. Se nenhum gerenciador de geometria for aplicado ao *widget*, ele existirá, mas não será visível ao utilizador. Neste caso o *widget* é chamado de virtual e serve, por exemplo, para criar uma GUI invisível que mesmo assim permita a atribuição de um evento a um procedimento.

Widgets

Um *widget* refere-se a um componente da interface gráfica (GUI - Graphic User Interface): um botão, uma caixa de texto, etc. De seguida, encontram-se alguns *widgets* que podem ser usados para o desenvolvimento da interface pretendida.

Button

O botão é um elemento da interface que permite ao utilizador realizar uma ação e interagir com a aplicação. A sua sintaxe no Tkinter é definida por:

```
widget = tk.Button(parent, option=value, ...)
```

```
```

onde:

- tk é a janela principal ou de topo, criada a partir da classe tk()

- parent é a janela pai do Button

Exemplo:
(Note a utilização do *pack* para tornar o *widget* visível.)

```
from tkinter import *
from tkinter.ttk import *

window = Tk()

window.title("Janela com Tkinter")
window.geometry('350x200')

btn = Button(window, text="Click Me")

btn.pack() #coloca o botão na janela principal

window.mainloop()

```

output

Entry

O *widget Entry* permite ao utilizador inserir uma linha de texto através da interface da aplicação. A sua sintaxe no Tkinter é definida por:

```
widget = tk.Entry(parent, option, ...)
```

Exemplo:

```
import tkinter as tk
from tkinter.ttk import *

window = tk.Tk()

window.title("Janela com Tkinter")
window.geometry('350x200')

ent = tk.Entry(window,width=30)

ent.pack() #coloca o entry na janela principal

window.mainloop()
```

output

Label

Este *widget* mostra uma ou mais linhas de texto, imagem ou mapa de bits. A sua sintaxe no Tkinter é definida por:

```
widget = tk.Label(parent, option, ...)
```

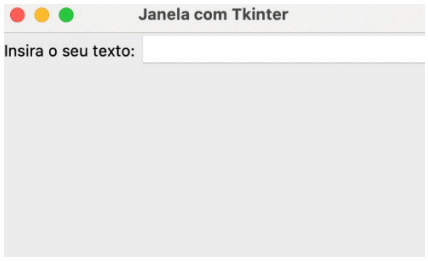
Exemplos:

```
1  import tkinter as tk
2  from tkinter.ttk import *
3
4  window = tk.Tk()
5
6  window.title("Janela com Tkinter")
7  window.geometry('350x200')
8
9  btn = tk.Label(window,text='Hello World')
10
11 btn.pack() #coloca o botão na janela principal
12
13 window.mainloop()
```

output


```
1  import tkinter as tk
2
3  window = tk.Tk()
4
5  window.title("Janela com Tkinter")
6  window.geometry('350x200')
7
8  lbl = tk.Label(window,text='Insira o seu texto:')
9  lbl.grid(column=0, row=0) #uso do grid para posicionar o elemento
10
11 ent = tk.Entry(window,width=30)
12 ent.grid(column=1, row=0)
13
14 window.mainloop()
```

output



Listbox

Este *widget* permite ao utilizador selecionar um ou mais itens de uma lista. O exemplo seguinte mostra a criação de uma Listbox.

```
1  import tkinter as tk
2
3  window = tk.Tk()
4
5  window.title("Janela com Tkinter")
6  window.geometry('350x200')
7
8  listbox_1 = tk.Listbox(window, selectmode=tk.MULTIPLE)
9  listbox_1.insert(1, 'Apple')
10 listbox_1.insert(2, 'Banana')
11 listbox_1.insert(3, 'Cherry')
12 listbox_1.insert(4, 'Mango')
13 listbox_1.insert(5, 'Orange')
14 listbox_1.pack()
15
16 submit = tk.Button(window, text='Submit')
17 submit.pack()
18
19 window.mainloop()
```

output

Scrollbar

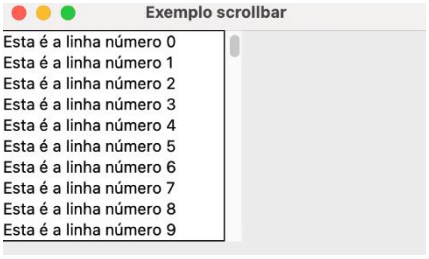
Este *widget* fornece um controlador de slide que é usado para implementar *widgets* de rolagem vertical e/ou horizontal, como Listbox, Text e Canvas. A sua sintaxe no Tkinter é definida por:

```
widget = Scrollbar (parent, option, ... )
```

Exemplo:

```
1  from tkinter import *
2
3  window = Tk()
4
5  window.title("Exemplo scrollbar")
6  window.geometry('350x200')
7
8  scrollbar = Scrollbar(window, orient=VERTICAL)
9
10 scrollbar.grid(row=0, column=1, sticky=NS) #orienta o scroll de norte(N) até sul(S)
11
12 listbox_1 = Listbox(window, selectmode=EXTENDED, yscrollcommand = scrollbar.set)
13
14 for line in range(100):
15    listbox_1.insert(END, "Esta é a linha número " + str(line))
16
17 listbox_1.grid(column=0, row=0)
18
19 scrollbar.config( command = listbox_1.yview )
20
21 window.mainloop()
```

output



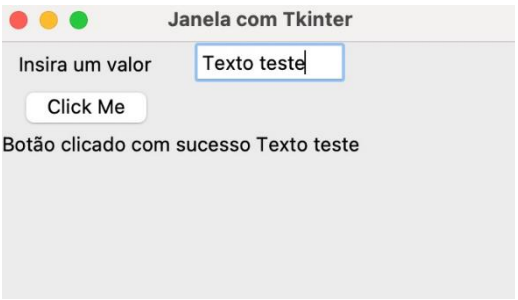
Eventos

Um clique do rato sobre a janela GUI, seja num *widget* ou num espaço vazio, ou o carregar de uma tecla do teclado, é chamado de evento. Os event handlers ("tratadores de eventos") são procedimentos programados para serem executados quando um evento específico acontece. Por exemplo, ao clicar num botão "Sair", a rotina que fechará a janela é o *event handler* correspondente a clicar nesse botão. Para que a janela seja efetivamente fechada ao clicar no botão "Sair", é necessário fazer *binding* do botão àquela rotina.

Exemplo:

```
1  from tkinter import *
2  window = Tk()
3  window.title("Janela com Tkinter")
4  window.geometry('350x200')
5
6  lbl = Label(window, text="Insira um valor")
7  lbl.grid(column=0, row=0)
8
9  txt = Entry(window,width=10)
10 txt.grid(column=1, row=0)
11
12 lbl2 = Label(window, text="")
13 lbl2.grid(column=0, row=3)
14
15 def clicked():
16     res = "Botão clicado com sucesso " + txt.get()
17     lbl2.configure(text = res)
18
19 btn = Button(window, text="Click Me", command=clicked)
20 btn.grid(column=0, row=2)
21
22 window.mainloop()
```

output



Conexão Python – MySQL

O Python pode ser usado para efetuar transações em bases de dados. No âmbito deste projeto, será necessário conectar o Python à base de dados MySQL criada na última aula de forma a povoar as tabelas com os dados preenchidos pelos utilizadores.

O primeiro passo, será a instalação do driver que o Python necessita para aceder à base de dados MySQL. Neste caso, o driver "MySQL Connector" pode ser instalado através do seguinte comando:

```
python -m pip install mysql-connector-python
```

O passo seguinte será criar a conexão à base de dados, usando o *username* e a *password* da base de dados MySQL previamente criada. De seguida, encontra-se um exemplo de uma conexão.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
print(mydb)
```

Após a conexão estar estabelecida, as operações transacionais da base de dados podem ser efetuadas. Em baixo, encontram-se os exemplos da execução de instruções de INSERT e SELECT. É importante ressalvar que, para que as transações do tipo INSERT, UPDATE e DELETE sejam efetivadas, é necessário finalizar as instruções com um commit.

Exemplos:

- INSERT

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
```

```
mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)

mydb.commit()

print(mycursor.rowcount, "record inserted.")
```

- SELECT

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```